An MVC-based Analysis of Object-Oriented System Prototyping for Banking Related GUI Applications

– Correlationship between OO Metrics and Efforts for Requirement Change –

Satoru Uehara, Osamu Mizuno, Yumi Itou and Tohru Kikuno
Department of Informatics and Mathematical Science,
Graduate School of Engineering Science, Osaka University, Japan
{s-uehara, o-mizuno, itou, kikuno}@ics.es.osaka-u.ac.jp

Abstract

In this paper we analyze statistically the efforts for C++ program modification which is needed by a given requirement change during the prototyping development of a certain GUI application. In the analysis we consider both C++ program P to be updated and the resultant C++ program P', and discuss the correlation between the values $\mathbf{M}(P)$ of the Object-Oriented metrics obtained from P and the efforts $\mathbf{E}(P')$ needed to produce P'. According to the definitions of $\mathbf{M}(P)$ and $\mathbf{E}(P')$, we present two approaches in this paper.

*In the first approach, we take $\mathbf{M}(P)$ as the value obtained by applying the metrics to the whole source code P and $\mathbf{E}(P')$ as the lines of codes(***LOC***) that are actually modified or created, respectively. However, the experimental result cannot show strong correlation between $\mathbf{M}(P)$ and $\mathbf{E}(P')$.*

Based on the analysis results of the first approach, we propose the second approach to consider the object-oriented properties more directly. The analysis process consists of following three steps: 1) take only the classes in P, classify them according to the MVC paradigm originally for Smalltalk their functions, and then evaluate $\mathbf{M}(P)$ for the classified classes, 2) in order to evaluate the efforts for code modification, define $\mathbf{E}(P')$ as the heuristic value which is calculated empirically from the numbers of methods and members created or modified, and 3) analyze the relationship between the values of $\mathbf{M}(P)$ on the classified classes in P and the values of $\mathbf{E}(P')$ on the source codes in P'. From the experimental result, we can prove there exists a high correlation between them.

1 Introduction

In order to manage the software development, a large number of methodologies and techniques have already been proposed. For realizing high quality and productivity of the software, the Object-Oriented(OO) paradigm has been attracted, and it is widely utilized into practice. As a matter of fact, the object-oriented paradigm has high capabilities to handle large systems, change them, reuse the part of systems, and so on, which are important factor to assure the quality of the product and the productivity of the development team.

Until now, various approaches have been presented as guidelines and actually unique graphical notations for deriving the object-oriented design, such as OMT[13] and Booch's method[2] have already been proposed. Additionally, a large number of object-oriented programming languages such as Smalltalk, C++, Objective-C and Java have been introduced. They of course implemented the object oriented properties such as class, encapsulation, inheritance, polymorphism, and so on. Thus they can make it easy to utilize the object-oriented paradigm into actual software development process. Recently, object-oriented approach is often used because it is very compatible with operating systems which have event driven architecture and Graphical User Interface(GUI), such as Windows.

On the other hand, in order to evaluate and support the OO development, the object-oriented metrics have been introduced, since the traditional software metrics were not appropriate for applying the OO development. The OO metrics are evaluated and newly proposed in much research[5, 7, 9].

While applying the OO paradigm into practical use, the prototyping development is more applicable rather than traditional waterfall model. The prototyping development can reveal the customer's requirement into the product, and it is compatible to the OO paradigm.

In the prototyping development, the first product is developed(including design, coding and test) based on the initial and thus incomplete requirement of the customer. Then the product is delivered to the customer and checked by them. The customer returns the changes in requirements to the developers. The developer performs the development to reflect the requirement. These process iterated until the customer satisfies the product or the deadline of delivery comes. Thus, it is

important to estimate the efforts to update the product for the prototyping development, when the changes in requirement are returned from the customer. The adequate estimation will make the prototyping development to be more productive one.

Therefore, we set the goal of our study as finding the practical method to estimate accurately the efforts for updating the program. However, in order to establish the estimating methodology, we must analyze the relationship among the developers, the environments, the source codes, the metrics, and so on. In this paper, we perform the analysis of the first step toward the final goal.

In this paper, we investigate the activities for updating the program due to changes of requirements. We consider the program P before updating and the program P' after updating, and investigate the correlation between the values of metrics $\mathbf{M}(P)$ which is measured from P, and the amounts of efforts $\mathbf{E}(P')$ which is calculated from P'. Based on the definitions of $\mathbf{M}(P)$ and $\mathbf{E}(P')$, we present two approaches in this paper.

In the first approach, we take $\mathbf{M}(P)$ as the value obtained by applying the OO metrics to the whole source code P and $\mathbf{E}(P')$ as the lines of codes(**LOC**) that are actually modified or created. However, the experimental result cannot show strong correlation between $\mathbf{M}(P)$ and $\mathbf{E}(P')$, since **LOC** does not consider the object-oriented characteristics and the application to the whole source code is too vague to conclude any specific properties.

Based on the results of the first approach, we propose the second approach to consider the object-oriented properties more directly. The key idea of second approach is as follows: 1) take the classes in P, classify them according to their functions using the MVC paradigm[4, 12], and then evaluate $\mathbf{M}(P)$ for the classified classes, 2) in order to evaluate the efforts for code modification, define $\mathbf{E}(P')$ as the heuristic value which is calculated empirically from the numbers of methods and members created or modified. We also consider the creation and modification of the classes separately.

Then we measure the values of $\mathbf{M}(P)$ for each classified classes Model, View and Controller. Finally we propose the formulas which can reflect the difficulties for updating. From the experimental result, we can prove there exists a high correlation between them.

This paper is organized as follows: Section 2 shows the motivation of our study. Section 3 explains the development process and the OO metrics which are used in this study. Section 4 shows the first analysis based on the legacy metric, and the result of it. Section 5 describes the key idea of this study and Section 6 shows the second analysis. Finally, Section 7 summarizes the main results and the future research work.

2 Motivation

2.1 Prototyping

Figure 1 shows an outline of typical prototyping process[6]. Generally speaking, the prototyping development proceeds consists of the rapid interactions between customers and developers. One of specific characteristics of the prototyping is there are not complete specifications during the development. The developers design and implement the product based on the customers' requirements, and deliver the prototype of program to the customers. The customers try to test the prototype, and return the changes of requirements to the developers. The loop is iterated until the customers satisfy the program.

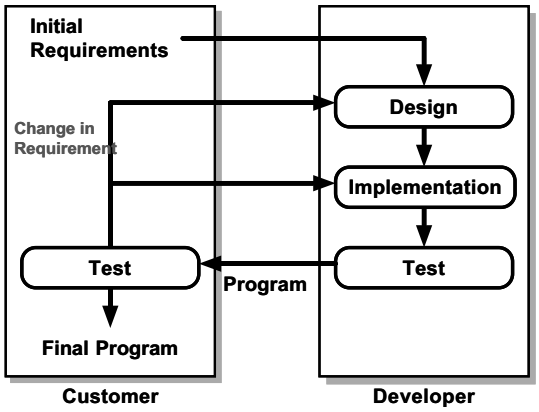

Figure 1. General prototyping development process

During the prototyping development, the productivity of development team is one of the most important factor, because the product have to be delivered to the customers as soon as possible. Thus, the estimation of the efforts on the prototyping development is an essential problem from the management point of view.

Generally speaking, it is very natual and easy to combine the prototyping development and object-oriented development. Because the object-oriented paradigm makes it easy to understand the system structure and reuse the previous components of other systems, and developers can deliver the product rapidly using the advantages of object-oriented development.

2.2 Efforts Estimation

Numerous studies have been done with the estimation on the software development process. Basili et al. have proposed the framework to predict the fault-proneness of the classes by using the object-oriented metrics[1, 3]. Since their aim is to estimate the quality, we cannot use their result directly to

our objective. On the other hand, Kusumoto et al. developed a simulator to estimate the efforts and faults of the software project[8]. However, their simulator was targeted to the standard waterfall model, so it is hard to apply to the prototyping development. Therefore we have to establish a new method to estimate the efforts for the object-oriented prototyping development.

Figure 2 shows a certain situation of the prototyping development. In this situation, we have a certain version of program P and requirement change δ from the customers, and we want to estimate the efforts needed to update the program P into P' according to δ.

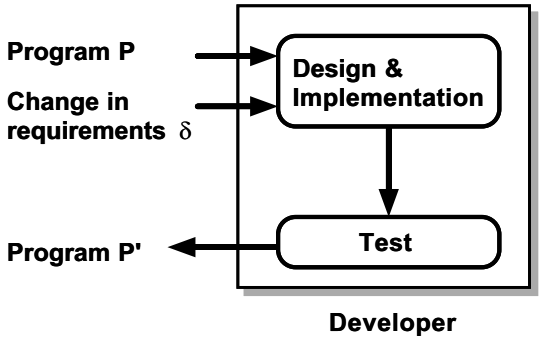

Figure 2. Estimation of the efforts

We can formulate this estimation as follows:

$$\mathit{Effort}(P \rightarrow P') = est(P, \delta, \varepsilon)$$

where ε is the environment factor such as developers, tools, and so on. In this study, we consider that both δ and ε are constant for simplicity. One of the reasons is that less complex method is required to use in the actual development field. Then the estimation formula is transformed as follows:

$$\mathit{Effort}(P \rightarrow P') = est(P)$$

We use the object-oriented metrics measured on program P to establish the function $est(P)$.

2.3 Objectives

As mentioned in the previous subsection, the final goal of our study is the estimation of the efforts. In order to establish the effort estimation method, the objective of this study is to analyze the relationship between the metrics measured before updating the program P and the efforts needed to update the program P into P'. For convenience, we denote the metrics measured on the program P as $\mathbf{M}(P)$, and the efforts needed to update the program P into P' as $\mathbf{E}(P')$. According to the definitions of $\mathbf{M}(P)$ and $\mathbf{E}(P')$, we present two approaches in this paper.

3 Process and Metrics

3.1 Software Development Process

Figure 3 shows the software development process actually adopted in the target project. The project is a development of banking related application. The platform of the program was Windows 95/NT, and development environment was Microsoft Visual C++. For the development, the OO development was performed according to the OMT.

Then we adopted a typical prototyping development process. Three developers were engaged in the initial design, and two of them took part in the successive prototyping development. During the prototyping, since the customers' requirement was changed twice, the development was iterated for three times. Finally, three versions of program P_1, P_2 and P_3 were generated.

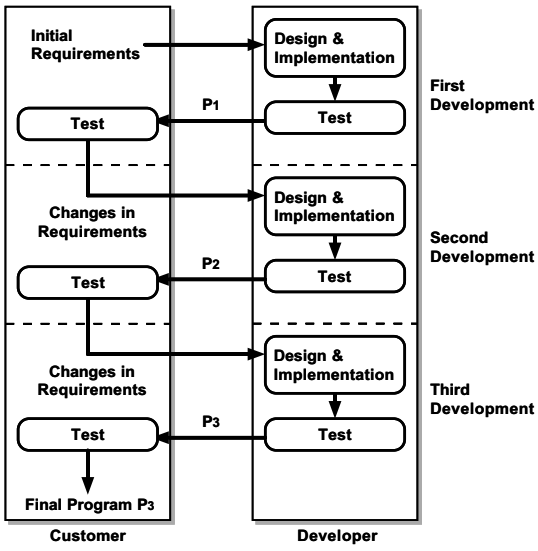

Figure 3. Development process

A change in requirement shown in Fig.3 consists of small requirement changes. For each small requirement change, the classes in the program are modified and/or created. Furthermore each change can be identified by the comments on the program.

3.2 Object-oriented metrics

In the object-oriented development, we can point out some factors which are related to the amount of efforts for updating the program.

- Complexity of the class definition.

  Generally, the more complex the class is, the more effort is needed to modify it. There are various metrics to

show the complexity of the class. Among them, we choose the weighted methods per class **WMC**, the depth of inheritance **DIT** and the number of children **NOC**.

- Interaction of the classes.

  Interacting the other object by message passing is one of important characteristics of the OO paradigm. However, if there are too many coupled classes in a class, it is difficult to modify the class because there are many classes which are affected by the modification. Thus we choose the metric **CBO**. Similarly, we consider that the interaction between the methods in the same class might affect the efforts. Thus we choose the metric **LCOM**.

In the following, we explain the definition of metrics to be used. These metrics are originally defined in [5]:

Table 1. Evaluation of metrics

Class	*WMC*	*DIT*	*NOC*	*CBO*	*LCOM*
C1	14	1	0	5	33
C2	11	1	0	2	37
C3	9	1	0	5	10
C4	21	1	0	4	140
C5	15	1	0	5	55
C6	5	1	0	3	4
C7	13	1	0	6	40
C8	6	1	0	3	9
C9	22	1	0	3	121
C10	11	1	0	6	27
C11	6	1	0	3	9
C12	21	1	0	3	144
C13	7	3	0	9	19
C14	3	4	0	2	3
C15	2	5	5	1	1
C16	5	6	0	2	10
C17	5	6	0	2	10
C18	5	6	0	2	10
C19	5	4	0	5	8
C20	7	5	0	4	21
C21	5	6	0	2	10
C22	5	6	0	2	10
C23	10	4	0	7	35
C24	9	4	0	8	20
C25	9	4	0	11	22
C26	8	4	0	8	14
C27	8	4	0	7	14
C28	15	4	0	15	73
C29	18	5	0	11	123
C30	17	4	0	14	64
C31	21	4	0	20	98
C32	15	4	0	12	75
C33	17	5	0	11	104
C34	11	6	0	9	55
C35	10	6	0	8	39
C36	17	4	0	20	88
C37	18	4	0	15	63
C38	10	6	0	8	39
C39	16	4	0	15	94
C40	17	4	0	13	64
C41	6	4	0	28	15
C42	39	3	0	19	619
C43	17	4	0	12	136

1) **WMC** (Weighted Methods per Class)

   WMC is the total complexity of weighted methods per class.

   Consider a class C with n methods $M_1, \cdots, M_n$ that are defined in the class. Let c_i be the complexity of a Method M_i:

$$\mathbf{WMC} = \sum_{i=1}^{n} c_i$$

   For simplicity, we assume that all the method complexities are unity ($c_i = 1$). Thus **WMC** actually shows the number of methods n.

2) **DIT** (Depth of Inheritance Tree)

   Depth of inheritance of the class is the **DIT** metric for the class:

$$\mathbf{DIT} = \text{the number of super-classes}$$

3) **NOC** (Number Of Children)

   NOC is a measure of how many subclasses are going to inherit the methods of the parent class:

$$\mathbf{NOC} = \text{the number of immediate subclasses subordinated to a class in the class hierarchy}$$

4) **CBO** (Coupling Between Object classes)

   CBO for a class is a count of the number of other classes to which it is coupled.

   Consider a class which has m member variables $A_1, \cdots, A_m$ and n methods $M_1, \cdots, M_n$. **CBO** can be expressed as follows:

$$\mathbf{CBO} = |\{O\} \cup \{R_1\} \cup \{R_2\} \cup \cdots \cup \{R_n\}|$$

   where $\{O\}$ is a set of classes which is defined as a type of member variables $A_1, \cdots, A_m$, and $\{R_i\}$ is a set of classes called by method M_i.

5) **LCOM** (Lack of COhesion in Methods)

   LCOM is the cohesion between methods.

   Consider a class C with n methods $M_1, \cdots M_n$. Let $\{I_i\}$ be a set of instance variables used by method M_i. Let $P = \{(I_i, I_j) | I_i \cap I_j = \phi\}$ and $Q = \{(I_i, I_j) | I_i \cap I_j \neq \phi\}$. Then **LCOM** can be expressed as follows:

$$\mathbf{LCOM} = \begin{cases} |P| - |Q| & \text{if } |P| > |Q| \\ 0 & \text{otherwise} \end{cases}$$

Then we apply the OO metrics to the program P. Table 1 shows the values of metrics for each class.

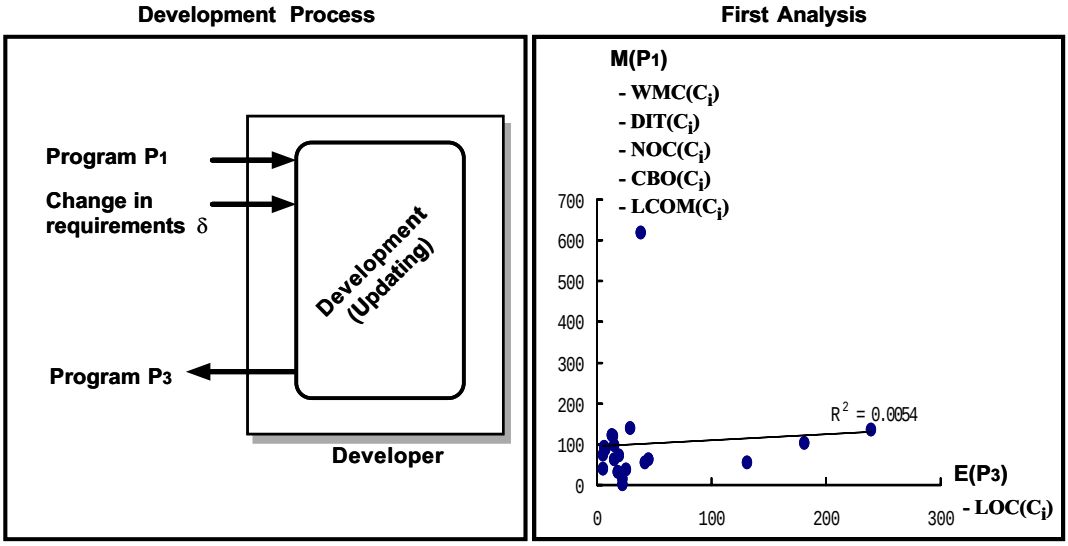

Figure 4. Outline of first analysis

3.3 Other data

Table 2 shows the resultant data of the target project. At first, the prototype program P_1 had 43 classes and the size was about 6.3**KLOC**[1]. After two successive requirement changes, the number of classes increases to 58, and the size becomes about 8.9**KLOC**.

Table 2. Characteristics of programs

Version	# of classes	*LOC*(P_i)
P1	43	6295
P2	53	7765
P3	58	8925

4 First Analysis

4.1 Outline

Figure 4 shows the outline of the first analysis. We analyze the correlation between the OO metrics $\mathbf{M}(P)$ and the efforts $\mathbf{E}(P')$. In the analysis below, for simplicity, we compound the development process by combining the second and third developments in Fig. 3 into one development.

Here, we introduce the notation $\mathbf{E}(C_i)$ to represent the efforts needed to modify the class C_i in P'. In the first analysis, we assume the efforts $\mathbf{E}(C_i)$ as the lines of codes which are modified and created $\mathbf{LOC}(C_i)$ in the modified class C_i. Then, we count $\mathbf{LOC}(C_i)$ for each class C_i in the program P':

$\mathbf{LOC}(C_i)$ = lines of codes which are modified or created in class C_i

Here, we use the notation $\mathbf{M}(C_i)$ to represent the OO metrics measured from the class C_i in the program P. Then we consider the correlation between $\mathbf{M}(C_i)$ and $\mathbf{E}(C_i)$. For example, Fig. 4 shows the correlation between the metric $\mathbf{LCOM}(C_i)$ and $\mathbf{LOC}(C_i)$.

4.2 Experimental Result

Table 3 shows the result of analysis. The correlation coefficients between the OO metrics(**WMC**, **DIT**, **NOC**, **CBO** and **LCOM**) and $\mathbf{LOC}(C_i)$, respectively, are shown in it.

Table 3. Correlation coefficients for $\mathbf{LOC}(C_i)$

	WMC	*DIT*	*NOC*	*CBO*	*LCOM*
Correlation coefficient	0.21	-0.02	0.05	0.12	0.07

As the result of analysis, we can say that there are not any specific high correlations between the OO metrics $\mathbf{M}(C_i)$ and $\mathbf{LOC}(C_i)$. Here, we discuss the reasons of the result. First, it is not adequate to consider **LOC** as efforts needed to modify the OO programs, because **LOC** cannot consider the OO characteristics in the program. Second, it can be said that the metrics obtained from the whole program were too

[1]Hereafter, the value of **LOC** implies the lines of code excluding the comments.

scattered to get any specific characteristics from them. In order to solve the problems found in the first analysis, we have to introduce some new idea.

5 Key Idea

5.1 Classifications of classes[4,12]

In order to classify the classes, we use the MVC paradigm[4, 12]. In the MVC paradigm, the user's input, the modeling of the external world, and the visual feedback to the user are explicitly separated as three types of functions: Model, View and Controller. The Model manages the behavior and data of the application domain, responds to requests for information about its state, and responds to instructions to change state. The View manages the graphical and/or textual output to the portion of the bitmapped display that is allocated to its application. Finally, the Controller interprets the mouse and keyboard inputs from the user, commanding the model and/or the view to change appropriately.

Since it is said that the MVC paradigm is suitable to the development of the GUI application, we apply it to our program and classify all the classes into three types. We expect that the properties of the classes are emphasized to extract the correlations between the OO metrics $\mathbf{M}(C_i)$ and the efforts $\mathbf{E}(C_i)$.

5.2 Empirically calculated efforts

From the result of the first analysis, we have to investigate the other efforts metric but **LOC**. Thus we introduce a new metric to measure the amount of efforts needed to update the programs. We consider two cases of updating the program: modification of the existing classes and creation of the new classes.

5.2.1 Formulas for efforts (modify)

In the case of modifying the existing class C_i, we calculate the efforts of modification as follows:

$$\mathbf{E}(C_i|modify) = w_1 \times mb_c + w_2 \times mt_m + w_3 \times mt_c$$

where mb_c is the number of member variables which are newly created on C_i, mt_m is the number of methods which are modified, mt_c is the number of methods which are newly created, and w_1, w_2 and w_3 are the weighting constants. In this study, we determine these constants as $(w_1, w_2, w_3) = (5, 1, 2)$. The weighting is currently based on the experience of developers. The developers in the target OO development consider that modifying a method is an easy work because it may not affect other parts of program. On the other hand, creating a method seems to be more difficult, since they have to refer the class definition or the other class. Additionally, creating a member variable could be the most difficult work, because if the type of new member variable is the other class, it is inevitable to refer the definition, structure and how to use.

5.2.2 Formulas for efforts (create)

In the case of creating a new class, we can measure the amount of creating efforts from the program P', but we cannot compare it with any metrics on the program P because a new class did not exist on P. However, it must not be ignored since the creation of the class is important factor of the OO development. Thus we try to introduce the substitute measure which can be compared with created new class.

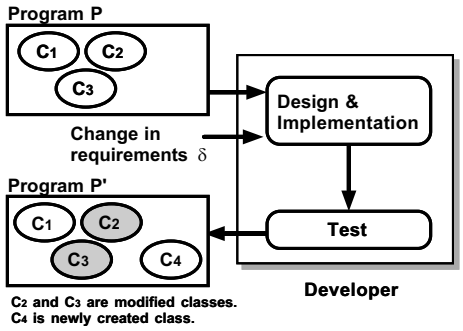

Figure 5. Related modified classes

In Fig. 5, consider a certain requirement change δ on the program P. In order to reflect the requirement change into P, we have to create a new class C_4 and modify two classes C_2 and C_3 at the same time. Here we define a set of C_2 and C_3 as the related classes with C_4 with respect to the change δ, and denote them as a set $rel\text{-}C_4$.

Then we define the $\mathbf{M}(rel\text{-}C_i)$, which denotes the OO metrics measured from the related classes in P.

$$\mathbf{M}(rel\text{-}C_i) = \sum_{C_j \ belong \ to \ rel\text{-}C_i} \mathbf{M}(C_j)$$

Then we calculate the efforts of creating the class C_i as follows:

$$\mathbf{E}(C_i|create) = w_1 \times mb_c + w_3 \times mt_c + w_1 \times rel\text{-}mb_c + w_2 \times rel\text{-}mt_m + w_3 \times rel\text{-}mt_c$$

where $rel\text{-}mb_c$ denotes the number of member variables which are newly created in the related classes $rel\text{-}C_i$, $rel\text{-}mt_m$ denotes the number of methods which are modified in $rel\text{-}C_i$, and $rel\text{-}mt_c$ denotes the number of methods which are created in $rel\text{-}C_i$.

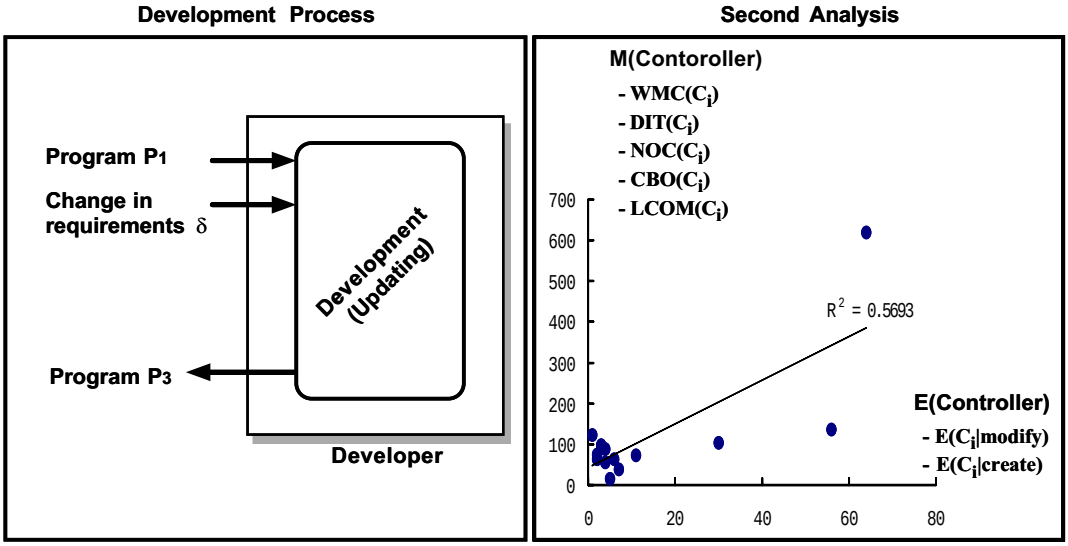

Figure 6. Outline of second analysis

5.2.3 Validation of formulas

In order to verify the validity of the formulas above, we analyze the correlation between the difficulty of the classes and the value of the proposed metric by the Spearman's rank correlation. Here we only show the validation of the formula for efforts of modification.

Table 4. Heuristic ranking of difficulty

Class	Rank	Class	Rank
C_1	12.5	C_{32}	18
C_4	5.5	C_{33}	3
C_5	2	C_{34}	7
C_7	19	C_{35}	10
C_9	12.5	C_{36}	14
C_{15}	15	C_{37}	5.5
C_{28}	9	C_{39}	17
C_{29}	16	C_{41}	20
C_{30}	11	C_{42}	4
C_{31}	8	C_{43}	1

First, we interviewed the developers of target project, and asked them to rank all the classes in program P by the difficulty to modify. Table 4 shows the ranking of difficulty for each modified class. Table 5, to be given in subsection 6.2, shows the caluculated efforts by the proposed weighting formula.

Then we perform the rank correlation analysis for the data in Table 4 and Table 5. As the result of analysis, the Spearman's rank correlation coefficient was 0.86. Thus, we can say that the proposed weighting formulae have a certain degree of validation.

6 Second Analysis

Based on the key idea shown in Section 5, we perform the second analysis.

6.1 Outline

Here, we explain the outline of the second analysis. In the analysis, we introduce two key factors. At first, we classify all the classes into three types: Model, View and Controller. In Table 1, the classes C_1 to C_{13} are Model classes, C_{14} to C_{27} and C_{28} to C_{43} are classified as View and Controller, respectively. Next, for each class, i.e. Controller, we investigate the correlation between the OO metrics $\mathbf{M}(Controller)$ and the efforts $\mathbf{E}(Controller)$. Figure 6 shows an example of the analysis for the Controller class, where we choose $\mathbf{E}(C_i|modify)$ for X axis and $\mathbf{LCOM}(C_i)$ for Y axis. The correlation coefficient was calculated as 0.75.

6.2 Experimental Result

Now we show the experimental results for two cases: modification of classes and creation of classes.

First, Table 5 shows the values of mb_c, mt_m, mt_c and calculated efforts $\mathbf{E}(C_i|modify)$ for each modified class.

Table 6 shows correlation coefficients between OO metrics $\mathbf{M}(C_i)$(**WMC**, **DIT**, **NOC**, **CBO** and **LCOM**,) and the efforts for modification $\mathbf{E}(C_i|modify)$. It is shown that the Controller class has high correlation with **WMC** and **LCOM**(the values of coefficients are 0.64 and 0.75, respectively). Figure 7 shows some typical scattered graphs of Controller class.

Table 5. Calculated efforts (Case of modification)

Class	mb_c	mt_m	mt_c	$E(C_i \mid modify)$
C1	1	1	3	12
C4	0	0	6	12
C5	3	6	15	51
C7	0	0	1	2
C9	1	1	2	10
C15	0	0	1	2
C28	1	2	1	9
C29	0	1	0	1
C30	0	1	2	5
C31	0	1	2	5
C32	0	0	1	2
C33	3	4	5	29
C34	0	4	0	4
C35	0	1	2	5
C36	0	2	1	4
C37	0	0	3	6
C39	0	1	1	3
C41	0	1	0	1
C42	12	0	4	68
C43	6	14	6	56

Table 6. Correlation coefficient for $\mathbf{E}(C_i|modify)$

	WMC	DIT	NOC	CBO	LCOM
Model	-0.14	----	----	0.04	-0.16
View	----	----	----	----	----
Controller	0.64	-0.36	----	0.00	0.75
All	0.53	-0.23	-0.15	0.00	0.64

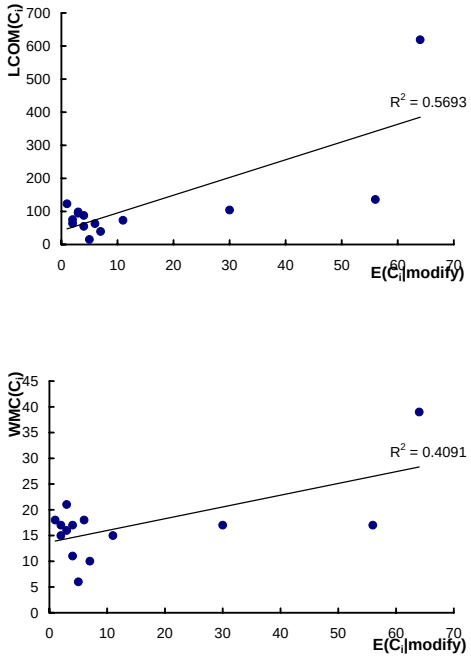

Figure 7. Scattered graph for the Controller (case of modification)

Next, we investigate the case of class creation. Table 7 shows the values of $rel\text{-}mb_c$, $rel\text{-}mt_m$, $rel\text{-}mt_c$ and calculated efforts $\mathbf{E}(C_i|create)$ for each created class.

Table 8 shows the correlation coefficients between the efforts of class creation $\mathbf{E}(C_i|create)$ and the OO metrics of related modified classes $\mathbf{M}(rel\text{-}C_i)$. It can be seen that relatively high correlations exist between them, especially for View and Controller classes. Figure 8 shows some typical scattered graphs for View class.

6.3 Discussions

From the result of the analyses, we can say that there are a certain extent of correlations between the OO metrics $\mathbf{M}(C_i)$ and the newly defined efforts of modification $\mathbf{E}(C_i)$. Especially, in case of creating classes, we can see relatively high correlations between them. For the classification of the classes, it can be seen that the Controller class has high tendency to correlate with the OO metrics, especially **WMC** and **LCOM**.

7 Conclusion

In this paper, we investigated the relationship between the OO metrics $\mathbf{M}(P)$ and the efforts $\mathbf{E}(P')$ for updating the program in the prototyping development. For the analysis, we made two approaches, and we can show some successful results in the second approach.

However, the result of the analysis might be a little bit limited. Especially, we must investigate the generalization of the weighting constant(currently determined heuristically) and the OO metrics itselves.

Considering the problems above, we summarize our future work as follows:

- We have to apply the analysis to other projects.
- The weighting formula shown in Section 5 have to be investigated its validity much more.
- For the choice of OO metrics, we have to examine the other metrics.
- Finally, we should establish the estimating method from $\mathbf{M}(P)$ to $\mathbf{E}(P')$.

References

[1] V. R. Basili, L. C. Briand and W. L. Melo: "A Validation of Object-Oriented Design Metrics as Quality

Table 7. Calculated efforts (Case of creation)

Class	$rel\text{-}mb_c$	$rel\text{-}mt_m$	$rel\text{-}mt_c$	mb_c	mt_c	$E(C_i \mid create)$
C44	4	13	8	1	4	52
C45	1	4	8	1	4	38
C46	1	3	7	1	4	35
C47	1	4	8	1	3	36
C48	4	3	8	1	3	50
C49	0	1	0	0	6	13
C50	0	0	2	1	3	15
C51	6	7	5	10	11	119
C52	1	4	5	4	7	53
C53	3	6	2	6	11	77
C54	6	7	5	5	9	90
C55	6	7	5	8	11	109
C56	4	3	8	3	17	88
C57	1	3	7	3	15	67
C58	1	0	5	1	13	46

Table 8. Correlation coefficient for $\mathbf{E}(C_i \mid create)$

	WMC	DIT	NOC	CBO	LCOM
Model	-0.35	----	----	0.35	-0.35
View	0.94	0.91	-0.45	0.97	0.90
Controller	0.87	-0.87	-0.87	0.87	0.87
All	0.82	0.78	-0.37	0.90	0.78

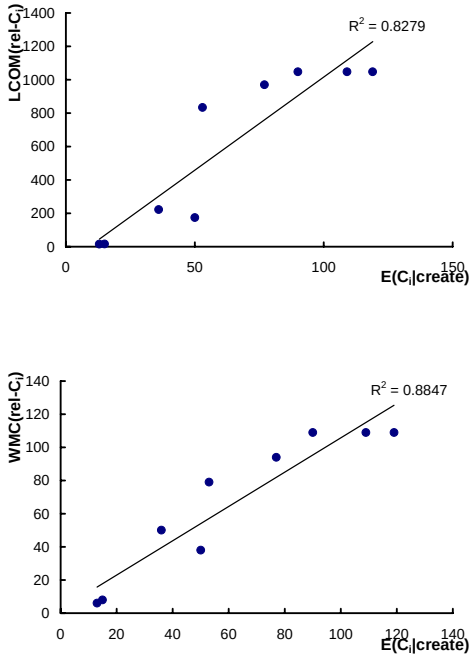

Figure 8. Scattered graph for the View (case of creation)

Indicators," IEEE Transactions of Software Engineering, Vol.22, No.10, pp.751–761, 1996.

[2] G. Booch: *Object Oriented Analysis and Design With Applications*, The Benjamin/Cummings, 1994.

[3] L. C. Briand, J. Daly, V. Porter and J. Wüst: "Predicting Fault-Prone Classes with Design Measures in Object-Oriented Systems," Proc. 9th International Symposium on Software Reliability Engineering(ISSRE'98), pp.334–343, 1998.

[4] S. Burbeck: "Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)," **http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html**, 1992.

[5] S. R. Chidamber and C. F. Kemerer: "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, Vol.20, No.6, pp.476–493, 1994.

[6] J. Martin: *Rapid Application Development*, Macmillan Publishing Company, 1991

[7] E. M. Kim: "Program Complexity Metric and safety Verification Method for Object-oriented Software Development," PhD. Dissertation, Osaka University, January, 1997.

[8] S. Kusumoto, O. Mizuno, Y. Hirayama, T. Kikuno, Y. Takagi and K. Sakamoto: "A New Project Simulator Based on Generalized Stochastic Petri-Net," Proc. 19th International Conference on Software Engineering(ICSE'97), pp.293–303, 1997.

[9] W. Li and S. Henry: "Object-oriented Metrics That Predict Maintainability," Journal of Systems and Software, Vol.23, pp.111–122, 1993.

[10] M. Lorenz and J. Kidd: *Object Oriented Software Metrics*, Prentice Hall, 1994.

[11] K. H. Möller and D. J. Paulish: *Software Metrics*, Chapman & Hall, 1993.

[12] L. J. Pinson and W. S. Wiener: *An Introduction to Object-Oriented Programming and Smalltalk*, Addison Wesley, 1988.

[13] J. Rumbaugh: *Object-Oriented Modeling and Design*, Prentice Hall, 1991.